ISBN 978-81-934174-2-3
6th International Conference on Engineering, Science, Technology and Industrial Applications (ESTIA-2017)
Istanbul (Turkey), September 08-10, 2017

Hybrid Hash-Based Join Algorithm for Parallel Computing

Amer Al-Badarneh, Yarub Wahsheh, Hassan Najadat

Jordan University of Science and Technology, Irbid, Jordan

Abstract**:** *In the last few years, smartphone devices becomes very popular. Increasing number of devices with internet connection, social networks and applications lead to large and complex data sets, more than ever before. Such data can be valuable source for analysis and simulation, in order to achieve better planning strategies and decision-making. Data sets need effective algorithms to capture, analyze, search, transfer, and manage, but challenges appear in treating big data. Classical database management systems, statistics and visualization tools cannot handle such amount of data sets, so there is a potential need to have effective algorithms that can handle these needs effectively in terms of time and size. In this study, we propose a parallel Hash-based join algorithm that aims to enhance the performance of join costly database operation. Our proposed algorithm supports both parallel and distributed computing, based on All Pair Partition and Broadcast Join algorithms. Experimental results compare the performance with All Pair Partition and shows valuable enhancements in terms of time.*

Keywords: *Join Algorithms, Parallel Computing, Hash-based Join, All-Pair Join, Bradcasting Join*

1. Introduction

With the increase of using smartphone devices and social networks, large and complex data sets are generated more than ever before. A new business trends related to data analysis becomes more popular. Scientists, traders, practitioners of medicine, advertising companies and governments are interested in retrieving beneficial information from big data analysis [1]. While traditional data processing techniques becomes inadequate, there is a potential need for new adapted methods that can analyze, capture, search, store, transfer and visualize "Big Data". An efficient and flexible way is required to get the needed services of analyzing, querying and extracting information [1].

Relational database management systems, statistics and visualization tools cannot handle such big amount of data sets. Instead of using classical methods, we need effectively processing algorithms [1]. Parallel and distributed software systems can play important roles in reducing time overhead of costly computer operations. Big data analysis and processing needs heavy select and join operations; an operation that combines columns from one or more database tables by using common values in each column [2]. However, this method is widely used in relational database models, creating new data sets that can be used in future operations. Join operations will lead in expensive cost in terms of CPU usage and I/O [2].

The MapReduce is a concept that provides a model for parallel and distributed data processing [3]. It was designed by Google in 2004 and implemented using Apache open-source project called Hadoop [3]. A typical MapReduce program contain two parts, Map () and Reduce (). In Map part, filtering and sorting functions take place, where in the Reduce part summary operations are applied. Although MapReduce offers valuable solution, join algorithms are not directly supported. Multiple approaches were designed to make usage in MapReduce to solve this problem using high-level languages [3].

Parallel computing is a principle used to solve large-scale problems; it is widely used in high-performance computing where many computations can be carried out at the same time. To solve a problem, multiple processing elements are used simultaneously, so that each element work on its part if the work. Processing

elements may include single processor computers, multi processors, several computers connected via network [4].

In this study we propose a hash based join algorithm that use hybrid technique between All Pairs Join and Broadcast Join. The proposed technique aims to get benefits of hash tables to enhance system performance; this technique can be used for both MapReduce and parallel computing. The rest of this study is divided as follows; Section 1 presents brief of related studies, Section 3 presents our proposed join algorithms, Section 4 presents experimental study, and finally Section 5 gives conclusion and future work.

2. Related Work

The study of [3] shows a comparative study of parallel and distributed join algorithms for MapReduce framework. The study describes the current used algorithms and divide them into these categories: Reduce side join, Map side join, semi Join, Range Practitioners and distributed cache.

However, the study shows a comparative analysis of algorithms with experiments about the execution time and processing the data skew.

The study of [5] shows interesting points about usage of MapReduce in join algorithms, the study describes strategies for Key-Value storage discussing theta join, All Pair Partition Joins, Repartition Join, Broadcasting Join, Semi Join and Per-Split Semi Join. The study shows examples of how these algorithms can work with listing advantages and disadvantages of each of them.

The study of [6] is a master thesis in which Join Algorithms using Map/Reduce were shown. The MapReduce environment and Hadoop implementation are described, also the current used techniques in join algorithms such as Reduce side join, Map side join and Broad cast join. The study extends these techniques and describes Multi-Way joins. Experiments were shown to evaluate these algorithms.

The study of [7] shows how MapReduce techniques can be used in index based join, to reduce amount of I/O and shuffling, the study shows an experiments that evaluate this algorithm, which shows good performance in terms of running time.

3. Proposed Technique

In this study, we propose a hybrid technique for two way join, our approach depends on All Pair Partition [5] and Broadcasting Join [5] and [6]. All Pair Partition Joins is a MapReduce technique used when more data is common between two data sets (R) and (L) [5]. If (R) table has |R| records, and (L) table has |L| records, then the product is a set of |R|*|L| records. Using MapReduce we can divide tables R and L into u and v portions respectively, and |R|*|L| can be calculated from u*v. Table 1 shows the portion part matrix with u*v size.

TABLE I: All Pair Partition Matrix with U*V Size

	V_1	V_2	V_3
U_1	Partition 1,1	Partition 1,2	Partition 1,3
U_2	Partition 2,1	Partition 2,2	Partition 2,3

Each Map task will deal with a single partition, results in output of <compound key, tagged record>. Compound key is the matrix key (row, column). To identify the original table of record, each record will have additional entry called tagged record, containing its original table name. Each group of data <compound key, tagged record> will be split into table R and L and then passed to the reducer, which will join data using the traditional joining method. Figure 1 shows all pairs partition joins.

Broadcasting Join described in both [5] and [6] seems interesting for enhancements. This technique is used when a table R is much smaller than table L. So all partitions of R can be transferred to the mappers, then loaded to memory in order to compute a hash table. For each record from a partition of table L, the map

function finds its reference in the hash table, and outputs only those it has referenced. All unreferenced records from table L will be ignored. Figure 2 shows broadcasting join example.

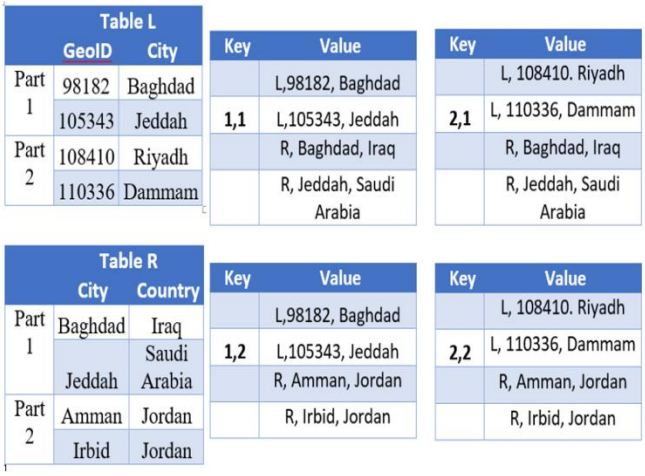

Fig. 1: All Pair Partition with equi-join for city field.

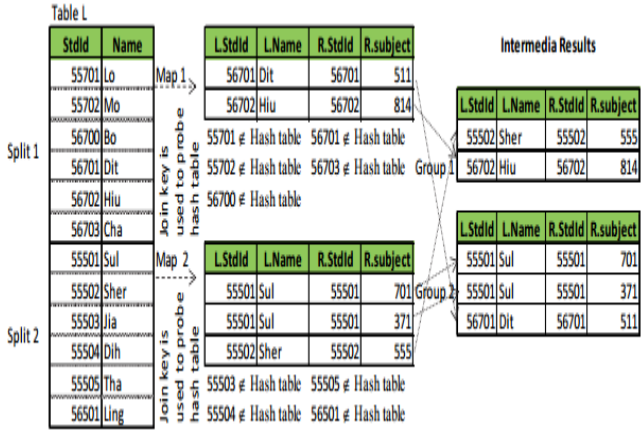

Fig. 2: Broadcasting join.

Our proposed hybrid system aims to get benefits of hash tables to reduce number of pairs needed for the join operation. Such approach can be applied on both MapReduce and parallel computing environments. Given two tables R and L; which have |R| and |L| records respectively, we will first apply a hash function for each of them. The function (ASCII (upper (left (join-key)))) mod N) +1, where N value equals the number of used threads. For example, if we want to use four threads (or four mappers in MapReduce) the hash function will divide our dataset to four groups. Each group will be assigned to a thread or mapper. Figure 3 shows how we divide table L into 4 groups (N=4).

Unlike Broadcasting Join, which applies the hash function to one table, we will apply the hash function on both tables R and L. This operation will divide each table into N groups, all tuples that have common left most values (start with the same character) will be grouped together. Instead of dividing tables L and R into (u*v) partitions, using hash-based divider can group common values together. So we will not need to make cross product (u*v) times. Using this operation we will have a thread of mapper that product each group in table L with its corresponding group on table R (instead of product each group of L with ALL groups of table R in traditional All Pair Join).

At the second phase, combine function (reduce) will combine all joined values at a single result file. For big data, all operations will be written directly to the disk, which means extra overhead in terms of I/O operations.

However, for our approach is dynamic and supports Broadcast Join if one of tables is small enough to be hashed and stored in the system memory. Figure 4 shows the algorithm flow chart.

TABLE L		
GeoID	City	HASH
57289	Hargeysa	1
71137	Sanaa	4
75441	Haddah	1
88319	Benghazi	3
90552	Sulaymaniyah	4
94787	Kirkuk	4
98182	Baghdad	3
99072	Mosul	2
104923	Khulays	4
323779	Antakya	2
601981	Vidsel	3
603035	Ranea	3
752713	Ahmediye	2

GROUP1		
GeoID	City	HASH
57289	Hargeysa	1
75441	Haddah	1

GROUP2		
GeoID	City	HASH
752713	Ahmediye	2
99072	Mosul	2
323779	Antakya	2

GROUP3		
GeoID	City	HASH
88319	Benghazi	3
98182	Baghdad	3
601981	Vidsel	3
603035	Ranea	3

GROUP4		
71137	Sanaa	4
90552	Sulaymaniyah	4
94787	Kirkuk	4
104923	Khulays	4

Fig. 3: Hash-Based division of table L to N groups

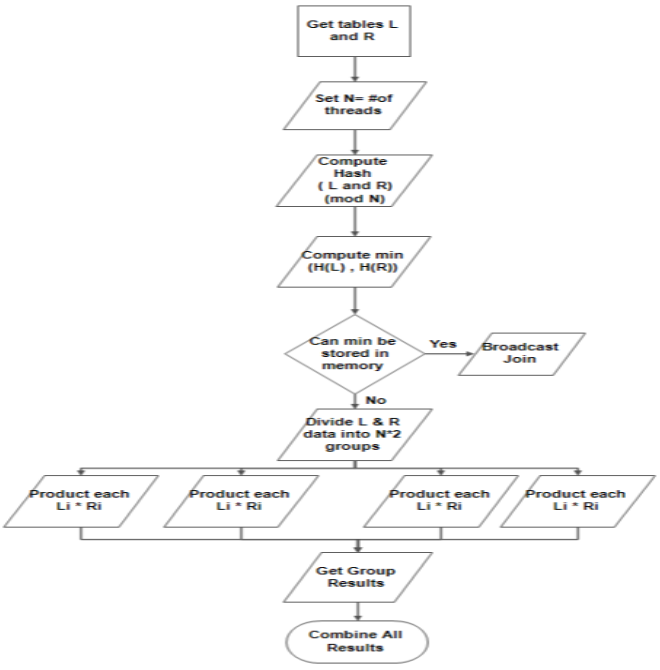

Fig. 4: Hybrid Hash-Based join flow diagram

As shown in Figure 4, N threads will be allocated to process two tables; L and R. Then a hash function is computed on both tables in order to divide them into smaller sets. Broadcast Join can be used once one table can fit in the available memory, but if these tables cannot fit into available memory, then these tables will be divided into N*2 groups. Note that each group will contain the common left most values (start with the same character), so similar data tuples will be grouped together. Then each group will make Li*Ri product (as a map phase) and finally results will be combined into single set.

4. Implementation and Evaluation

In order to evaluate the algorithm, it was implemented using Java programming language. The implementation specifications are:

- Ubuntu Linux 15.10 operating system.
- Intel core i5 processor.
- 8 GB of memory.
- One master thread and five workers.

The implementation follows these steps:

1. The master thread (thread 1) calls thread 2, and each of them divide one table and compute hashes.
2. Division groups are written as files.
3. The master thread call workers (4 threads) to deal with data groups.
4. Each worker works on two groups Li and Ri, as All Pair Partitions but for two groups only.
5. Results are written to the results file.

Figures 5 and 6 shows the pseudo code of our proposed algorithm.

```
Job 1: Generate a file with hash values from join key attribute
    Class Master (Thread1)
        Read_File (Table L)
            Call Thread2
            While join key & value (k, v) in all relations do
                N ← Number of threads;
                C ← get left most character;
                V ← ASCII(C);
                H ← (V mod N) +1;
                File ← File (H, 1);
                File ← Write (k, v)
            End While
    Class Thread2
        Read_File (Table R)
            While join key & value (k, v) in all relations do
                N ← Number of threads;
                C ← get left most character;
                V ← ASCII(C);
                H ← (V mod N) +1;
                File ← File (H, 2);
                File ← Write (k, v)
            End While
```

Fig. 5: Generate hash values.

```
Class Thread (i)
    Read_File (File (i, 1))
        While (k, v) [1] in all relations do
            K1← key1;
            Read_File (File (i, 2))
            While (k, v) [2] in all relations do
                K2← key2;
                If (K1==K2) Then
                    Write (k, v) [1] (k, v) [2] to Results
                End IF
            End While
        End While
```

Fig. 6: Join thread.

In order to evaluate the performance of our algorithm, we have implemented All Pair Partition described in [5]. Applying these two algorithms on World Cities Dataset [8]; a dataset of cites in the world, with additional information about GeoID, Country and Continent. Note that the size is 96394 records. We choose City Name attribute as the join key between two tables L<City Name, GeoID> and R<City Name, Continent Name, Country Name>.

We did not apply Broadcast Join on this dataset, because of its huge size. We have applied both our proposed algorithm and All Pair Partition. With the same results, these two algorithms were able to apply join process; results showed that our proposed algorithm shows better results in terms of delay. Finishing its work

with total delay of 5 minutes and 9 seconds. While All Pair Partition finished with 15 minutes and 36 seconds, our measured enhancement ratio is about 33.1%. Delay comparison is shown in Figure 7.

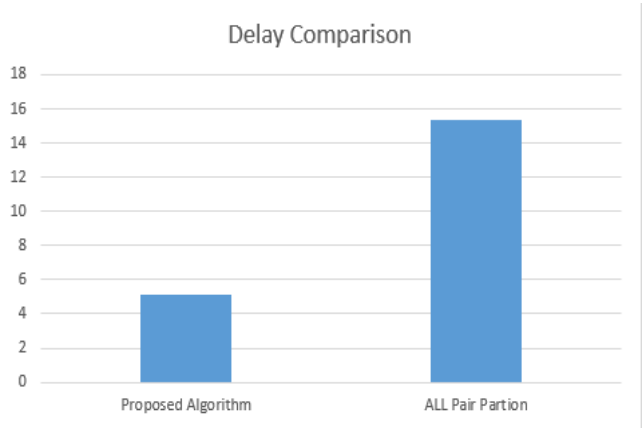

Fig. 7: Measured delay results.

Results showed that there are important differences between the two studied algorithms. However, there are still disadvantages and the algorithm needs to be utilized more. Table 2 shows the advantages and disadvantages of our proposed algorithm.

TABLE II: Advantages and Disadvantages of the Proposed Technique

Advantages	Disadvantages
Simple and easy to implement	Need time to compute hashes for each table.
Can be used for both parallel computing and MapReduce	Size overhead
Can support network data transfer instead of using files	Disk I/O overhead (for parallel computing)
Support Broadcast Join	Needs more utilization

5. Conclusion and Future Work

In this paper, we have proposed a Hash-Based join algorithm that aims to take advantages of both All Pair Join and Broadcasting Join algorithms. Since Join operations lead in expensive cost in terms of CPU usage and I/O, our proposed technique aims to reduce the needed time of applying such operations. The proposed algorithm works on two way join, and explores how parallel and distributed computing can be utilized to enhance the performance of current techniques. Experimental results showed that the algorithm can utilize All Pair Join algorithm with approximately 33.1% enhancement ratio.

As a future work, we plan to compare this algorithm with other techniques, such as broadcasting join, utilize the implementation, try other datasets and implement distributed version that use network data transfer.

6. References

[1] "Big data", *Wikipedia*, 2017. Available: en.wikipedia.org/wiki/Big_data. [Accessed: 15-June-2017].

[2] "Join_(SQL)", *Wikipedia*, 2017. Available: en.wikipedia.org/wiki/Join_(SQL). [Accessed: 15-May-2017].

[3] A. Pigul, "Comparative Study Parallel Join Algorithms for MapReduce environment," in *Proc. of ISP RAS*, 2012, pp. 285-306.

https://doi.org/10.15514/ISPRAS-2012-23-17

[4] "Parallel Computing", *Wikipedia*, 2017. Available: en.wikipedia.org/wiki/Parallel_computing. [Accessed: 12- May-2017].

[5] D. Van Hieu, et al., ”MapReduce join strategies for key-value storage,” in *Proc. of 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 2014.
https://doi.org/10.1109/JCSSE.2014.6841861

[6] J. Chandar, “Join algorithms using map/reduce” Magisterarb. University of Edinburgh, 2010.

[7] M. Khafagy, “Indexed Map-Reduce Join Algorithm,” *Int. Journal of Sci. & Eng. Research*, vol. 6, pp. 705-711, 2015.

"Free World Cities Database | MaxMind", *Maxmind.com*, 2017. [Online]. Available: www.maxmind.com/en/free-world-cities-database. [Accessed: 8-January-2017].